Static reduction of two dimensional finite element models

20/05/13

Contents

1 Introduction

Here, the theory and implementation of a procedure of sub-structuring or static reduction is described, which can be applied to two dimensional finite element models. The implementation of the static reduction procedure is carried out using MATLAB and much of it assumes that the finite element model has been created using the commercial finite element software package ABAQUS/CAE. This just happens to the software used by the author, however equivalent procedures could be performed on models made in other finite element packages, and could be implemented using a difference software package as well.

At the current level of sophistication, this procedure applies only to two dimensional, quasi-static, linear elastic analyses, in which body forces are not present. Further, this procedure currently is restricted to cases in which the contact area lies approximately along a straight line. This permits the analysis of, for example, the face of a square in contact with a larger block, contact between slightly curved surfaces that can be approximated using half-plane theory, or a body with multiple parallel contact interfaces, such that the normal and tangential directions do not vary along the contact or from one contact to another. In its current form, this method cannot be used to analyse cases in which the normal and tangential directions vary along the interface or from one interface to another, such as is the case for shrink-fit shafts, pin-in-hole contact, contacts with multiple non-parallel interfaces, contacts with highly curved interfaces, and so on. However, further development of this method should allow several of these restrictions to be relaxed.

2 Example ABAQUS Model: Elastic on Rigid Contact

In order to illustrate how to perform the static reduction procedure, it is helpful to consider a very simple example model. Thus, instructions are provided that detail how to create an extremely simple two dimensional, planar finite element model in ABAQUS of a single square block. The model that is created in ABAQUS is simply a square with no loads or constraints applied to it. The global stiffness matrix from this model is then extracted and used to create the reduced contact stiffness matrix that corresponds to the case of frictional contact between the square's bottom face and a rigid plane obstacle, as shown in Figure 1. Note that, the *Input file*

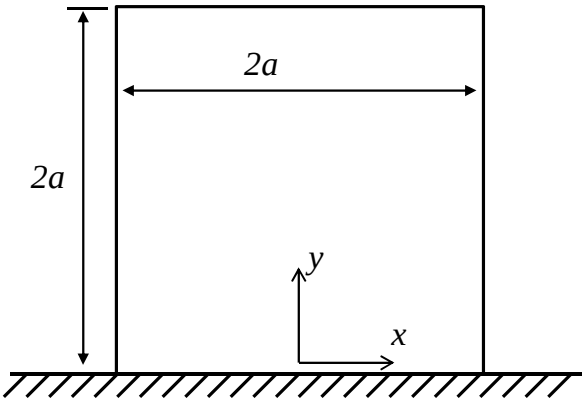

Figure 1: A diagram of a square in contact with a rigid plane obstacle along its bottom face, showing the coordinate system adopted in the ABAQUS model and static reduction procedure. Note that for this model $a = 1$.

produced by completing the steps in § 2.1, is provided in full in § A.1. If the reader is already very familiar with ABAQUS, then he or she may wish simply to review the *Input file* in § A.1, and then skip ahead to § 3.

2.1 Create a simple example model in ABAQUS

The process described in this section assumes that the reader has some basic familiarity with ABAQUS. If the reader is unfamiliar with ABAQUS, please quickly review § 2.2.1 of the *ABAQUS/CAE User's Manual*, which gives the names of the various menus bars etc. that will be referred to in this section. The user's manual can be accessed from within ABAQUS by clicking `Help` → `Search & Browse Manuals...`,[1] and then clicking the first link under *Modelling and Visualization*, which is *ABAQUS/CAE User's Manual*.

In order to make a basic model from which to extract the global stiffness matrix, the reader should perform the following steps:

1. First, open ABAQUS. Check that the `Part` *Module* is selected on the *Context Bar*. To create a new *Model*, click `File` → `New Model Database` → `With Standard/Explicit Model` from the *Menu Bar*.

2. To make a new *Part*, click `Part` → `Create...` from the *Menu Bar*. In the window that appears, select the following options: `Name: RR_example, Modelling Space: 2D Planar, Type: Deformable, Base Feature: Shell, Approximate Size: 10`. Then, click `Continue...`.

3. To draw the *Part*, click `Add` → `Line` → `Rectangle` from the *Menu Bar*. For the starting coordinate type `-1,1` and press `Enter`. Then type `1,-1` for the ending coordinate, and then press `Enter`. Then press the `Esc` key once to exit the `Rectangle` function, and click the `Done` button at the bottom left of the *Viewport*.

4. To create a *Material*, switch from the `Part` *Module* to the `Property` *Module* on the *Context Bar*. Click `Material` → `Create...` from the *Menu Bar*, and in the

[1] Note that, `A` → `B` means for the reader to select option `B` from drop-down menu `A`.

window that appears, click Mechanical → Elasticity → Elastic, and input the following values: Young's Modulus: 2e+11, Poission's Ratio: 0.3, and then click OK.

5. To create a *Section*, click Section → Create... from the *Menu Bar*. In the window that appears, select the options Name: Section-1, Category: Solid, Type: Homogeneous, and then click Continue.... In the window that appears, select the option Material: Material-1, leave the box Plane stress/strain thickness unchecked, and click OK.

6. To *Assign* the *Section* to the *Part*, click Assign → Section from the *Menu Bar*, then click on the part in the *Viewport*, and then click the Done button at the bottom of the *Viewport*. In the window that appears, select the options: Section: Section-1, Assignment: From Section, and click OK.

7. To *Mesh* the *Part*, switch from the Property *Module* to the Mesh *Module* on the *Context Bar*. Then click Seed → Part... from the *Menu Bar*, and, in the window that appears, input the value Approximate global size: 1, while leaving all other options at their default value, and click OK. Then click Mesh → Part... from the *Menu Bar* and press Enter.

8. To make three *Node Sets* (internal, contact, and externally loaded nodes), first, open the *Set Manager* by clicking Tools → Set → Manager... from the *Menu Bar*. In the window that appears, click Create..., and in the subsequent window that appears, select the options: Name: aa-contact, Type: Node, and click Continue.... Then, select[2] the three nodes along the bottom face of the square, and then click the Done button at the bottom of the *Viewport*. Repeat this process for the top three nodes naming the set b-ext_loaded, and, again, for the middle three nodes naming the set c-internal.[3] Once these three *Node Sets* have been created, click Dismiss on the *Set Manager* window. A diagram of the ABAQUS model showing the three *Node sets*, where the black circles represent individual nodes and the grey ellipses show the nodes to be included in each *Node set* is shown in Figure 2 .

9. To create an *Instance* in the *Assembly*, first, switch from the Mesh *Module* to the Assembly *Module* on the *Context Bar*. Then, click Instance → Create... from the *Menu Bar*, and in the window that appears, click OK.

10. To create a *Step*, switch from the Assembly *Module* to the Step *Module* on the *Context Bar*. Then, click Step → Create... from the *Menu Bar*, and in the window that appears, select the options: Name: Step-1, Insert new step after: Initial, Procedure type: General, and, in the box directly below, in the same window, select the option Static, General, and then click Continue.... Then, in the window that appears, press OK.

[2]Nodes may be selected either by holding the Shift key and selecting the nodes individually, or by clicking and dragging mouse such that the selection field captures the desired nodes. If, for some reason, undesired nodes are selected, they may be removed from the selection by holding the Ctrl key and clicking on the nodes to be deselected.

[3]Note that *Node sets* appear in the *Input file* in the order in which they are created, not in alphabetical order as they appear in ABAQUS.

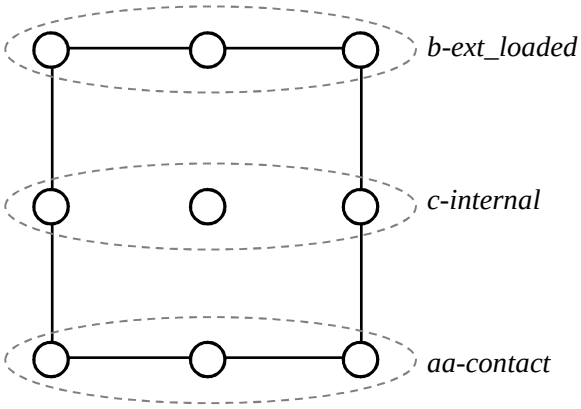

Figure 2: A diagram of the ABAQUS model, which shows the three *Node sets*, where the black circles represent individual nodes and the grey ellipses show the nodes to be included in each *Node set*.

11. To create a *Job*, switch from the `Step` *Module* to the `Job` *Module* on the *Context Bar*. Then, click `Job` → `Create...` from the *Menu Bar*, and select the options: `Name: example, Source: Model` and, in the box below, select `Model-1`, and press `Continue...`. In the box that appears, leave all of the options at their default values, and press `OK`.

12. To create an *Input file* from the *Job*, click `Job` → `Manager...` from the *Menu Bar*, and, in the window that appears, select `example`, and click the `Write Input` button, and then click `Dismiss` to exit the *Job Manager*.

Step 12 will write the *Input file* to the folder where ABAQUS stores all of its *Job* files and data when running *Jobs*. The default folder is usually the `C:\Temp` folder. The file will be named `example.inp`, however, this type of file can be read with a plain text (`.txt`) editor, such as `Notepad`. A copy of the *Input file* that should be produced when the above steps are performed is provided in § A.1.

Completing steps 1 - 12 should produce an *Input file* that can be run in ABAQUS without errors. Please check that this is the case before proceeding to §2.2. To test that the *Job* runs without errors, open the *Job Manager*, as in step 12, by clicking `Job` → `Manager...` from the *Menu Bar*, and then *Submit* the *Job* by clicking `Submit`. Upon doing this, a warning box may appear, but just click `OK`. Then, click `Monitor` in the *Job Manager*. The *Job* should be *Completed* successfully after a few moments, and without errors. If this is not the case, please review the steps 1 - 12. Once the *Job* has *Completed* successfully, click `Dismiss` on any windows that remain open.

2.2 Run an ABAQUS job from an *Input file*

To run a *Job* from an *Input file*, switch to the `Job` *Module* on the *Context Bar*. Then, to create the *Job*, click `Job` → `Create...` from the *Menu Bar*, and select the option `Source: Input file`. Then, click on the button to the right of the text `Input file:` and browse the files to locate the *Input file* that was created in § 2.1. Once located, select that file, and then click `OK`. Now, open the *Job Manager* by clicking `Job` → `Manager...` from the *Menu Bar*, and, in the window that appears, select `example-1`. Check that the *Model* field of the

selected *Job* says `File:example.inp`, and not `Model-1`, and then click `Submit` and then `Monitor` to check that the *Job* runs successfully.

3 Transfer data from ABAQUS to MATLAB

In this section, the process of exporting data from the ABAQUS finite element model, including the global stiffness matrix and node sets, and importing this data into MATLAB for processing is described.

3.1 Modify the *Input file* and export the global stiffness matrix

In order to perform the static reduction procedure, the global stiffness matrix must be extracted from the ABAQUS finite element model. To do this, the *Input file* must be modified.[4] This may be done by locating the *Input file*, which is usually located in the `C:\Temp` folder, and opening it with a standard text editor (`.txt` file extension).[5] Once the *Input file* has been located and opened, use the text editor's find utility (usually the key combination `Ctrl+f`), and search for the string of code shown directly below.

```
** STEP: Step-1
```

Directly above this line of code, some additional lines of code, with the command to export the stiffness matrix, must be added to the *Input file*. For ABAQUS version 6.11, the following code must be added.

```
*STEP, name=exportmatrix
*MATRIX GENERATE, STIFFNESS
*MATRIX OUTPUT, STIFFNESS, FORMAT=MATRIX INPUT
*END STEP
**
```

For versions of ABAQUS prior to 6.11, the code below must be added.

```
*STEP, name=exportmatrix
*MATRIX GENERATE, STIFFNESS
*END STEP
**
```

Once the *Input file* is modified as described above, the global stiffness matrix can be exported by running the modified *Input file* in ABAQUS, as described in § 2.2. Once the *Job* has completed, the global stiffness matrix should appear in ABAQUS's *Job* folder (which is, again, usually the `C:\Temp` folder). If this procedure is performed on the *Input file* provided in § A.1, then the resulting `.mtx` file should appear in the `C:\Temp` folder as the file `example-1_STIF1.mtx`. Note that, the `.mtx` file format is ABAQUS's matrix output format, and that this format can be read in a normal plain text editor (`.txt` file extension).

[4] For an example *Input file* see § A.1, which contains the a copy of the *Input file* created in § 2.1.

[5] The author recommends the program `Notepad++`, due to several useful features that are very helpful when processing *Input files*. `Notepad++` is free and can be downloaded at `http://notepad-plus-plus.org`.

3.2 Convention for numbering degrees of freedom in MATLAB

For two dimensional models, each node in the ABAQUS model is associated with two degrees of freedom. To refer to an individual node in the ABAQUS finite element model according to the node number assigned to it in ABAQUS, the shorthand 'A-node' is used. For the purposes of the static reduction procedure, it is easier to work with individual degrees of freedom than with A-nodes, which have multiple degrees of freedom. Therefore, a new convention is adopted in MATLAB in which each degree of freedom is assigned a number, and is treated as an individual node. To refer to an individual degree of freedom, when numbered according to this new convention, the shorthand 'M-node' is used. The conversion from A-nodes to M-nodes is performed as

$$\text{M-node} = 2(\text{A-node} - 1) + \text{DOF} \tag{1}$$

where 'DOF' is the degree of freedom of the A-node that the M-node represents. Thus, for two dimensional problems, such that DOF $\in \{1, 2\}$, there are two M-nodes corresponding to each A-node. Also note that, as a result of this convention, *odd* numbered M-nodes correspond to degree of freedom 1, and *even* numbered M-nodes correspond to degree of freedom 2.

3.3 Import the ABAQUS global stiffness matrix into MATLAB

Given that the static reduction procedure to be described is to be carried out using the MATLAB software package, the first step is to convert the `.mtx` file, which contains the ABAQUS finite element model's global stiffness matrix, into MATLAB's sparse matrix format. ABAQUS's sparse matrix format consists of R rows and 5 columns, where R is the number of non-zero stiffness values in the global stiffness matrix. The 5 columns of the sparse matrix in the `.mtx` file are denoted using the letters a, b, c, d, k, and the quantity contained in each column is listed in Table 1. Each row of the `.mtx` file specifies the stiffness value k, between degree of freedom b of A-node a, and degree of freedom d of A-node c, where $a, c \in \{1, ..., M\}$, where M is the total number of A-nodes in the finite element model, and where $b, d \in \{1, 2\}$ for two dimensional models.

The global stiffness matrix contained in the `.mtx` file is, first, to be read in MATLAB using the function `dmlread`. This $R \times 5$ matrix is then to be used to compose a $2M \times 2M$ sparse matrix in which the value of each element in the matrix is the stiffness value k, between two M-nodes. The row and column numbers corresponding to the each stiffness value represent the two M-nodes that the stiffness value connects, i.e. the row number of the stiffness value corresponds to the **single** M-node that is represented by the first *and* second columns of the `.mtx` file, and the column number corresponds to the **single** M-node that is represented by the third *and* fourth columns of the `.mtx` file.[6]

[6] In the global stiffness matrix, stiffness values are specified between individual degrees of freedom. Thus, using the A-node notation, both the A-node number and the degree of freedom must be specified. This is the reason that the two columns in the `.mtx` file correspond to only one M-node (or degree of freedom).

Column	1	2	3	4	5
Symbol	a	b	c	d	k
Quantity	A-node	DOF of a	A-node	DOF of c	Stiffness value

Table 1: A table showing the quantity that is contained in each column of the `.mtx` file.

3.3.1 Input parameter and example usage

MATLAB code is provided in § A.2 for a function named `import_stiffness_matrix`. This function imports the data contained in a `.mtx` file into $2M \times 2M$ sparse matrix. The only input parameter that must be input into this function is the file path of the `.mtx` file, e.g. `example-1_STIF1.mtx`. The global stiffness matrix is output as a $2M \times 2M$ sparse matrix, and is this function's only output variable. For example, if both the matrix file `example-1_STIF1.mtx` and the function file `import_stiffness_matrix.m` are contained in MATLAB's working directory, then the following line of code will import the global stiffness matrix into the MATLAB variable `K`.

```
K = import_stiffness_matrix('example-1_STIF1.mtx');
```

3.4 Create *Node sets* in ABAQUS

To perform the static reduction, the M-nodes must first be categorised as being internal, externally loaded, or along a contact interface. Although it is possible to categorise the M-nodes manually in MATLAB for very simple models, the purpose of this static reduction procedure is to increase the efficiency of processing models of significant complexity. In these cases, it is essential to have an automated process for categorising the M-nodes. To this end, *Node sets* are to be created in ABAQUS, which then appear in the *Input file*. For complex models, it is often better to create many *Node sets* in ABAQUS that group A-nodes related to a particular feature together, e.g. A-nodes corresponding to a particular external load, or boundary condition. These various sets of A-nodes can then converted to sets of M-nodes in MATLAB, which can then be concatenated into sets of contact, externally loaded, or internal M-nodes.

3.5 Import the ABAQUS *Node sets* into MATLAB

The *Node sets* that are created in ABAQUS, as described in § 3.4, are listed in the ABAQUS *Input file*. Therefore, these *Node sets* can be imported into MATLAB by reading this file. MATLAB code is provided in § A.3 for a function named `import_node_set`, which reads an ABAQUS *Input file*, searches for the *Node set* that is specified, and imports the *Node set* into MATLAB. This function outputs two MATLAB variables, which are both row vectors. The first output variable is a row vector containing the M-nodes in the *Node set* that is specified, and the second output variable is a row vector containing the A-nodes.

3.5.1 Basic input parameters

The function `import_node_set` imports one node set at a time. The first input parameter of this function is the file path of the *Input file*, e.g. `'RR_example.inp'` (assuming that this file is in MATLAB's working directory). The second input parameter is the name of the *Node set* that is to be imported, e.g. `'c-internal'`. The third input parameter is the number of lines contained in the ABAQUS *Input file*. The number of lines in the *Input file* can be determined by opening the *Input file* in a text editor and finding the line number corresponding to the last line in the file. Alternatively, this value can be set to some arbitrarily large value that is certainly larger than the number of lines in the *Input file*, e.g. 500,000. (The purpose of this input is to prevent the function from becoming caught in an infinite loop if it cannot find the specified *Node set* in the *Input file*.)

If the M-nodes corresponding to both degrees of freedom from each A-node are to be imported from the *Node set*, and the order of the nodes in the *Node set* is not important, then this function can be called with only these three input parameters. This situation is likely to arise when importing the internal nodes, for example.

3.5.2 Importing M-nodes of only one direction

In some cases, however, only M-nodes corresponding to a particular direction (i.e. 1 or 2) from each A-node in the *Node set* are to be imported. This situation arises when displacement boundary conditions are applied to a group of nodes within ABAQUS that restrict the nodes' motion. If some direction of motion is restricted, then the M-nodes corresponding to this direction are to be completely excluded from the static reduction. These M-nodes are to be excluded because, by definition, they are not degrees of freedom. It follows that if both degrees of freedom of a given A-node are restricted within the ABAQUS model, then both M-nodes corresponding to this A-node are to be completely excluded from the static reduction procedure.

In cases when only M-nodes of a particular direction are to be imported into MATLAB, a fourth input parameter can be entered into the function `import_node_set`. The options for this parameter are `1`, `2`, or `'both'`. This parameter specifies which degree of freedom is to be imported into the row vector of M-nodes (the first output variable of this function). This input parameter only affects the first output variable (the M-nodes), and has no influence on the second output variable (the A-nodes). If `1` is input, then only M-nodes of degree of freedom 1 are imported, and similarly if `2` is input. If `'both'` is input, then both M-nodes are imported from each A-node in the *Node set*. This is the default option (i.e. `'both'`), and is the option that is assumed if only the first three inputs are specified.

In the case described above in which one of the directions of motion of a group of A-nodes is restricted, only the direction that is not restricted is to be imported into MATLAB. This can be achieved by inputting either `1` or `2` in the fourth input position of the function, depending on whether A-node's displacement in direction 1 or 2 is **not** restricted, respectively.

3.5.3 Importing M-nodes of each direction separately

In some cases, the two M-nodes corresponding to one A-node may belong to different categories. Suppose, for example, that a displacement-controlled load is to be applied to a group of nodes in direction 1, but that their displacement in direction 2 is not restricted or controlled. In this situation, the M-nodes corresponding to direction 1 are to be imported and included in the set of externally loaded nodes, while the M-nodes corresponding to direction 2 are to be imported and included in the set of internal nodes. In situations such as this, in which the two M-nodes corresponding to each A-node in the *Node set* belong to different categories, the *Node set* must be imported into MATLAB twice. First, the *Node set* must be imported with the fourth input option as `1`. Next, the *Node set* must be imported again, except this time with the fourth input set as `2`. These two sets of M-nodes can then be included in the static reduction as different categories of nodes.

3.5.4 Sorting a *Node Set*

For some *Node sets* the nodes must be listed in a particular order, e.g. the set of contact nodes. Input positions five and six of the function `import_node_set` enable the user to sort the nodes in the *Node set* that is to be imported according to the nodes' spatial coordinates in the

finite element model. The fifth input parameter can be input either as `1` or `2`, and specifies the coordinate direction that the nodes are to be sorted with respect to. The sixth input parameter can be input either as `'ascending'` or `'descending'`, and specifies whether the nodes are to be sorted in ascending or descending order.

Note that if a *Node set* is to be sorted (and parameters are input in the fifth and sixth positions), then the fourth input parameter cannot be left blank, and must be specified either as `1`, `2`, or `'both'`. Also note that sorting the *Node set* uses more computational resources than does importing a *Node set* without sorting it. Therefore, for the sake of computational efficiency, *Node sets* should only be sorted when it is strictly necessary, which it always is for the contact nodes and sometimes is for the externally loaded nodes. It is worth mentioning that *Node sets* containing internal nodes should not be sorted.

3.5.5 Examples

To illustrate the use of the function `import_node_set`, several examples are considered, which use the *Input file* from the example model created in § 2 for demonstration. In all the examples below, it is assumed that this *Input file* and a file named `import_node_set.m`, which contains the function `import_node_set`, are in MATLAB's working directory.

The most simple case of importing a *Node set* arises when importing A-nodes that do not have any boundary conditions applied to them in ABAQUS, are not along a contact interface, and are not loaded in any way. In such a case, both M-nodes from each A-node are to be imported, and this is done in the following MATLAB code.

```
[Mnodes_internal, Anodes_internal] = ...
    import_node_set('example.inp', 'c-internal', 72);
```

Suppose, now, that the top three nodes of the example model are to have a uniform external displacement applied to them in the vertical direction (i.e. direction 2), but that direction 1 of these nodes is to be left free. In this case, the M-nodes in direction 1 are to be included in the static reduction as internal nodes, while the M-nodes of direction 2 are to be included as externally loaded nodes. Also, because the load that is to be applied is uniform, the order of the nodes in the *Node set* is not important. Therefore, these nodes can be imported using the following MATLAB code.

```
[Mnodes_top_internal, Anodes_top_internal] = ...
    import_node_set('example.inp', 'b-ext_loaded', 72, 1);
[Mnodes_ext_loaded, Anodes_ext_loaded] = ...
    import_node_set('example.inp', 'b-ext_loaded', 72, 2);
```

Finally, the contact nodes must be loaded into MATLAB. Suppose that the contact nodes are to be ordered from left to right. This corresponds to increasing (i.e. ascending) values of direction 1. The code below imports the contact nodes in this order.

```
[Mnodes_contact, Anodes_contact] = ...
    import_node_set('example.inp', 'aa-contact', 72, 'both', ...
    1, 'ascending');
```

These four *Node sets* can then be grouped according to their categories.

```
contact_Mnodes = Mnodes_contact;
ext_loaded_Mnodes = Mnodes_ext_loaded;
internal_Mnodes = [Mnodes_internal, Mnodes_top_internal];
```

4 Static Reduction Procedure: Theory

In this section, the mathematical and theoretical basis of the static reduction procedure is described. First, the case of a single elastic body in contact with a rigid surface is considered, which is then extended to the case of contact between two elastic bodies.

4.1 Sign Convention

Given that, in its current form, this static reduction procedure applies only to two dimensional problems, there are two degrees of freedom at each A-node (see § 3.2). Therefore, for an individual A-node along the contact interface j, the nodal force $\boldsymbol{f}_j^C$, and displacement $\boldsymbol{u}_j^C$, can be written as

$$\boldsymbol{f}_j^C = \left\{ \begin{array}{c} q_j \\ p_j \end{array} \right\} \tag{2a}$$

$$\boldsymbol{u}_j^C = \left\{ \begin{array}{c} v_j \\ w_j \end{array} \right\}, \tag{2b}$$

where, if a right hand x, y coordinate set is defined at each A-node with the positive y-direction pointing towards the interior of the body, then p_j, q_j are defined as positive in the positive x, y directions, respectively, and similarly for v_j, w_j. (A diagram of this coordinate system is shown in Figure 1.) Thus, the force normal to the contact surface is considered positive when compressive, and the normal displacement along the contact interface is similarly taken to be positive towards the interior of the body.[7] These two element, column vectors, which contain the nodal forces or displacements for each individual contact A-node j, can then be used to create a column vector containing the full set of contact forces

$$\boldsymbol{f}^C = \left\{ \boldsymbol{f}_1^C, \boldsymbol{f}_2^C, \boldsymbol{f}_3^C, ..., \boldsymbol{f}_N^C \right\}^T \tag{3a}$$

$$= \left\{ q_1, p_1, q_2, p_2, q_3, p_3, ..., q_N, p_N \right\}^T, \tag{3b}$$

and contact displacements

$$\boldsymbol{u}^C = \left\{ \boldsymbol{u}_1^C, \boldsymbol{u}_2^C, \boldsymbol{u}_3^C, ..., \boldsymbol{u}_N^C \right\}^T \tag{4a}$$

$$= \left\{ v_1, w_1, v_2, w_2, v_3, w_3, ..., v_N, w_N \right\}^T, \tag{4b}$$

where N is the number of contact A-nodes. The relationship between contact forces and displacements can now be written as

$$\boldsymbol{f}^C = \boldsymbol{f}^w + \boldsymbol{K}^C \boldsymbol{u}^C, \tag{5}$$

where $\boldsymbol{K}^C$ is the reduced contact stiffness matrix, which is positive definite and symmetric, and $\boldsymbol{f}^w$ is a column vector, with its elements organised in a similar fashion to $\boldsymbol{f}^C$, that corresponds to the nodal forces that would arise as a result of an externally applied load if the displacements

[7] Note that, the ABAQUS model created in § 2.1 uses ABAQUS's default X-Y global coordinate set, where the positive Y-direction points up, and the positive X-direction points to the right. A square is drawn and the contact A-nodes are taken to be along the bottom face. Thus, for this model, the positive Y-direction in ABAQUS's global coordinate system corresponds to the positive p_j and w_j directions, and the positive X-direction in ABAQUS corresponds to the positive q_j and v_j directions.

at the contact A-nodes were constrained to be zero, i.e. $\boldsymbol{u}^C = 0$.

4.2 Eliminate the internal degrees of freedom

The force displacement relationship for the full stiffness matrix of a single elastic body in contact with a rigid surface can be written as

$$\boldsymbol{f} = \boldsymbol{K}\boldsymbol{u}, \tag{6}$$

where $\boldsymbol{f}$ is a row vector of all nodal forces, $\boldsymbol{u}$ is a row vector of all nodal displacements, and $\boldsymbol{K}$ is the global stiffness matrix. To reduce the global stiffness matrix $\boldsymbol{K}$, by eliminating the internal degrees of freedom, first, the global stiffness matrix $\boldsymbol{K}$, is to be re-partitioned into several sub-matrices. This is much more easily done using the notation of M-nodes than A-nodes. Therefore, these sub-matrices are created by grouping the stiffness values between various combinations of internal, externally loaded, and contact M-nodes, such that

$$\begin{Bmatrix} \boldsymbol{f}^I \\ \boldsymbol{f}^E \\ \boldsymbol{f}^C \end{Bmatrix} = \begin{bmatrix} \boldsymbol{K}^{II} & \boldsymbol{K}^{IE} & \boldsymbol{K}^{IC} \\ \boldsymbol{K}^{EI} & \boldsymbol{K}^{EE} & \boldsymbol{K}^{EC} \\ \boldsymbol{K}^{CI} & \boldsymbol{K}^{CE} & \boldsymbol{K}^{CC} \end{bmatrix} \begin{Bmatrix} \boldsymbol{u}^I \\ \boldsymbol{u}^E \\ \boldsymbol{u}^C \end{Bmatrix}, \tag{7}$$

where the superscripts I, E, C correspond to the internal, externally loaded, and contact M-nodes, respectively. Thus, $\boldsymbol{f}^I, \boldsymbol{f}^E, \boldsymbol{f}^C$ denote the nodal forces at the internal, externally loaded, and contact M-nodes, respectively, and similarly for the nodal displacements $\boldsymbol{u}^I, \boldsymbol{u}^E, \boldsymbol{u}^C$. In a similar fashion, the global stiffness matrix $\boldsymbol{K}$, is partitioned into several sub-matrices corresponding to the stiffness values between the M-nodes belonging to these three categories. For example, $\boldsymbol{K}^{II}$ corresponds to the stiffness values between internal M-nodes and other internal M-nodes, $\boldsymbol{K}^{EI}$ to the values between externally loaded M-nodes and internal M-nodes, $\boldsymbol{K}^{CE}$ to the values between contact M-nodes and externally loaded M-nodes, and so on.

Because only cases in which body forces are not present are to be considered, the force at any pair of adjacent internal M-nodes must sum to zero, i.e. $\boldsymbol{f}^I = 0$. The nodal forces must also sum to zero at M-nodes along the outer surface of the body unless the M-nodes are: externally loaded, along the contact interface, or fixed in ABAQUS. Therefore, M-nodes not falling into these categories are to be categorised as internal M-nodes. Note that, M-nodes that have boundary conditions applied to them in ABAQUS that restrict their motion are to be completely excluded from the reduction process.[8]

By making use of the condition that $\boldsymbol{f}^I = 0$, equation (7) can be simplified. First, consider that the first row of equation (7) can be written as

$$\boldsymbol{f}^I = \boldsymbol{K}^{II}\boldsymbol{u}^I + \boldsymbol{K}^{IE}\boldsymbol{u}^E + \boldsymbol{K}^{IC}\boldsymbol{u}^C, \tag{8}$$

and the second two rows may be written as

$$\begin{Bmatrix} \boldsymbol{f}^E \\ \boldsymbol{f}^C \end{Bmatrix} = \begin{bmatrix} \boldsymbol{K}^{EI} & \boldsymbol{K}^{EE} & \boldsymbol{K}^{EC} \\ \boldsymbol{K}^{CI} & \boldsymbol{K}^{CE} & \boldsymbol{K}^{CC} \end{bmatrix} \begin{Bmatrix} \boldsymbol{u}^I \\ \boldsymbol{u}^E \\ \boldsymbol{u}^C \end{Bmatrix}. \tag{9}$$

[8]M-nodes that are fixed in ABAQUS are to be completely excluded from the reduction process because, by definition, if their motion is restricted, they are not degrees of freedom.

By setting $\boldsymbol{f}^I = 0$ in equation (8), $\boldsymbol{u}^I$ can be solved for as

$$\begin{aligned} 0 &= \boldsymbol{K}^{II}\boldsymbol{u}^I + \boldsymbol{K}^{IE}\boldsymbol{u}^E + \boldsymbol{K}^{IC}\boldsymbol{u}^C \\ \boldsymbol{K}^{II}\boldsymbol{u}^I &= -\boldsymbol{K}^{IE}\boldsymbol{u}^E - \boldsymbol{K}^{IC}\boldsymbol{u}^C \\ \boldsymbol{K}^{II}\boldsymbol{u}^I &= -\begin{bmatrix}\boldsymbol{K}^{IE} & \boldsymbol{K}^{IC}\end{bmatrix}\begin{Bmatrix}\boldsymbol{u}^E \\ \boldsymbol{u}^C\end{Bmatrix} \\ \boldsymbol{u}^I &= -\left[\boldsymbol{K}^{II}\right]^{-1}\begin{bmatrix}\boldsymbol{K}^{IE} & \boldsymbol{K}^{IC}\end{bmatrix}\begin{Bmatrix}\boldsymbol{u}^E \\ \boldsymbol{u}^C\end{Bmatrix}. \end{aligned} \tag{10}$$

To eliminate $\boldsymbol{u}^I$, equation (10) can be substituted into equation (9) as

$$\begin{aligned} \begin{Bmatrix}\boldsymbol{f}^E \\ \boldsymbol{f}^C\end{Bmatrix} &= \begin{bmatrix}\boldsymbol{K}^{EI} & \boldsymbol{K}^{EE} & \boldsymbol{K}^{EC} \\ \boldsymbol{K}^{CI} & \boldsymbol{K}^{CE} & \boldsymbol{K}^{CC}\end{bmatrix}\begin{Bmatrix}\boldsymbol{u}^I \\ \boldsymbol{u}^E \\ \boldsymbol{u}^C\end{Bmatrix} \\ &= \begin{bmatrix}\boldsymbol{K}^{EE} & \boldsymbol{K}^{EC} \\ \boldsymbol{K}^{CE} & \boldsymbol{K}^{CC}\end{bmatrix}\begin{Bmatrix}\boldsymbol{u}^E \\ \boldsymbol{u}^C\end{Bmatrix} + \begin{bmatrix}\boldsymbol{K}^{EI} \\ \boldsymbol{K}^{CI}\end{bmatrix}\boldsymbol{u}^I \\ &= \left[\begin{bmatrix}\boldsymbol{K}^{EE} & \boldsymbol{K}^{EC} \\ \boldsymbol{K}^{CE} & \boldsymbol{K}^{CC}\end{bmatrix} - \begin{bmatrix}\boldsymbol{K}^{EI} \\ \boldsymbol{K}^{CI}\end{bmatrix}\left[\boldsymbol{K}^{II}\right]^{-1}\begin{bmatrix}\boldsymbol{K}^{IE} & \boldsymbol{K}^{IC}\end{bmatrix}\right]\begin{Bmatrix}\boldsymbol{u}^E \\ \boldsymbol{u}^C\end{Bmatrix}. \end{aligned} \tag{11}$$

It is now convenient to define a new matrix

$$\boldsymbol{L} = \left[\begin{bmatrix}\boldsymbol{K}^{EE} & \boldsymbol{K}^{EC} \\ \boldsymbol{K}^{CE} & \boldsymbol{K}^{CC}\end{bmatrix} - \begin{bmatrix}\boldsymbol{K}^{EI} \\ \boldsymbol{K}^{CI}\end{bmatrix}\left[\boldsymbol{K}^{II}\right]^{-1}\begin{bmatrix}\boldsymbol{K}^{IE} & \boldsymbol{K}^{IC}\end{bmatrix}\right], \tag{12}$$

and to partition it as

$$\begin{Bmatrix}\boldsymbol{f}^E \\ \boldsymbol{f}^C\end{Bmatrix} = \begin{bmatrix}\boldsymbol{L}^{EE} & \boldsymbol{L}^{EC} \\ \boldsymbol{L}^{CE} & \boldsymbol{L}^{CC}\end{bmatrix}\begin{Bmatrix}\boldsymbol{u}^E \\ \boldsymbol{u}^C\end{Bmatrix}. \tag{13}$$

4.3 Determine the reduced stiffness matrix and the load vectors

In order to calculate the reduced contact stiffness matrix $\boldsymbol{K}^C$, and the load vector $\boldsymbol{f}^w$, it must first be determined whether the external loads are applied in force or displacement control. It is important to emphasise that the reduced stiffness matrix $\boldsymbol{K}^C$, is affected by whether the external loads are applied in force control or displacement control. This is because the reduced contact stiffness matrix is the solution of a problem in which the prescribed loading conditions are replaced by equivalent homogeneous conditions. These conditions are: traction-free in the force controlled case, and zero displacement in the displacement controlled case.

4.3.1 Force control

For the case when the external load is applied in force control, the distribution of force at the nodes along which the load is applied $\boldsymbol{f}^E$, is known. The first row of equation (13) can be written out and used to solve for $\boldsymbol{u}^E$ as

$$\begin{aligned} \boldsymbol{f}^E &= \boldsymbol{L}^{EE}\boldsymbol{u}^E + \boldsymbol{L}^{EC}\boldsymbol{u}^C \\ \boldsymbol{L}^{EE}\boldsymbol{u}^E &= \boldsymbol{f}^E - \boldsymbol{L}^{EC}\boldsymbol{u}^C \\ \boldsymbol{u}^E &= \left[\boldsymbol{L}^{EE}\right]^{-1}\left[\boldsymbol{f}^E - \boldsymbol{L}^{EC}\boldsymbol{u}^C\right]. \end{aligned} \tag{14}$$

Then, writing out the second row of equation (13) and substituting in equation (14) gives

$$\begin{aligned}\boldsymbol{f}^C &= \boldsymbol{L}^{CE}\boldsymbol{u}^E + \boldsymbol{L}^{CC}\boldsymbol{u}^C \\ &= \boldsymbol{L}^{CE}\left[\left[\boldsymbol{L}^{EE}\right]^{-1}\left[\boldsymbol{f}^E - \boldsymbol{L}^{EC}\boldsymbol{u}^C\right]\right] + \boldsymbol{L}^{CC}\boldsymbol{u}^C \\ &= \boldsymbol{L}^{CE}\left[\boldsymbol{L}^{EE}\right]^{-1}\boldsymbol{f}^E + \left[\boldsymbol{L}^{CC} - \boldsymbol{L}^{CE}\left[\boldsymbol{L}^{EE}\right]^{-1}\boldsymbol{L}^{EC}\right]\boldsymbol{u}^C, \end{aligned} \tag{15}$$

which can be written in the form given in equation (5), which is reproduced here for convenience

$$\boldsymbol{f}^C = \boldsymbol{f}^w + \boldsymbol{K}^C\boldsymbol{u}^C,$$

where

$$\boldsymbol{f}^w = \boldsymbol{K}^E\boldsymbol{f}^E, \tag{16}$$

and

$$\boldsymbol{K}^E = \boldsymbol{L}^{CE}\left[\boldsymbol{L}^{EE}\right]^{-1} \tag{17a}$$

$$\boldsymbol{K}^C = \left[\boldsymbol{L}^{CC} - \boldsymbol{L}^{CE}\left[\boldsymbol{L}^{EE}\right]^{-1}\boldsymbol{L}^{EC}\right]. \tag{17b}$$

4.3.2 Displacement control

For the case when the external load is applied in displacement control, the displacement of the nodes along which the load is applied $\boldsymbol{u}^E$, is known. The second row of equation (13) can be written out as

$$\boldsymbol{f}^C = \boldsymbol{L}^{CE}\boldsymbol{u}^E + \boldsymbol{L}^{CC}\boldsymbol{u}^C, \tag{18}$$

where equation (5) still applies, but where

$$\boldsymbol{f}^w = \boldsymbol{K}^E\boldsymbol{u}^E, \tag{19}$$

and

$$\boldsymbol{K}^E = \boldsymbol{L}^{CE} \tag{20a}$$

$$\boldsymbol{K}^C = \boldsymbol{L}^{CC}. \tag{20b}$$

4.4 Contact of two elastic bodies

To extend this procedure to the case of two elastic bodies in contact, first, a separate finite element model must be created for each body, the global stiffness matrix of each individual model must then be extracted, and the reduced contact stiffness matrix $\boldsymbol{K}^C$, as well as all the loading vectors $\boldsymbol{f}^w$, must be computed. Care must also be taken to ensure that in each separate model there are an equal number of contact A-nodes, that the contact A-nodes are distributed in the same way along both bodies, and that the two sets of contact A-nodes are arranged in the same sequence. Note that, the reduced contact stiffness matrix of a body with a rigid body mode is singular. For most load cases, one body is fixed in space, while the other has rigid body degrees of freedom. The matrix manipulations carried out below, which combine the reduced stiffness matrices of two elastic bodies, require that at least one of the reduced contact stiffness matrices be **non-singular**. Therefore, body 2 is designated as the body that does **not** have any

rigid body degrees of freedom. If, neither of the two bodies have a rigid body mode, then which body is designated 1 or 2 may be decided arbitrarily.

For the case of elastic-elastic contact considered here, the sign convention described in § 4.1 is adopted for each body separately. Therefore, along the contact interface p_j^1 and p_j^2 are both positive when compressive (and point towards the interior of the respective body), where the superscripts $1, 2$ denote the body to which the component pertains. Thus, when the two bodies are in contact, Newton's third law requires that $p_j^1 = p_j^2$ and $q_j^1 = q_j^2$, which is equivalent to the statement $\boldsymbol{f}^C = \boldsymbol{f}_1^C = \boldsymbol{f}_2^C$, where the subscripts $1, 2$ denote the body to which the vector of contact force relates.[9] As a result of this sign convention, subscripts are unnecessary for the vector of contact forces, and the relationship between forces and displacements can be written as

$$\boldsymbol{f}^C = \boldsymbol{f}_1^w + \boldsymbol{K}_1^C \boldsymbol{u}_1^C \tag{21a}$$

$$\boldsymbol{f}^C = \boldsymbol{f}_2^w + \boldsymbol{K}_2^C \boldsymbol{u}_2^C, \tag{21b}$$

where the subscripts $1, 2$ denote the body to which $\boldsymbol{f}^w$, $\boldsymbol{K}^C$, or $\boldsymbol{u}^C$ relate.

The contact displacements must satisfy the conditions

$$v_j = v_j^1 + v_j^2 \tag{22a}$$

$$w_j = w_j^1 + w_j^2, \tag{22b}$$

where the superscripts for v_j, w_j are assigned in a similar fashion as for q_j, p_j, which is equivalent to the condition

$$\boldsymbol{u}^C = \boldsymbol{u}_1^C + \boldsymbol{u}_2^C. \tag{23}$$

In order to merge equations (21a) and (21b), first, equation (21b) can be solved for $\boldsymbol{u}_2^C$ as

$$\begin{aligned} \boldsymbol{f}^C &= \boldsymbol{f}_2^w + \boldsymbol{K}_2^C \boldsymbol{u}_2^C \\ \boldsymbol{K}_2^C \boldsymbol{u}_2^C &= \boldsymbol{f}^C - \boldsymbol{f}_2^w \\ \boldsymbol{u}_2^C &= \left[\boldsymbol{K}_2^C\right]^{-1} \left(\boldsymbol{f}^C - \boldsymbol{f}_2^w\right). \end{aligned} \tag{24}$$

Then, equation (24) can be substituted into equation (23), and solved for $\boldsymbol{u}_1^C$

$$\begin{aligned} \boldsymbol{u}^C &= \boldsymbol{u}_1^C + \boldsymbol{u}_2^C \\ \boldsymbol{u}^C &= \boldsymbol{u}_1^C + \left[\boldsymbol{K}_2^C\right]^{-1} \left(\boldsymbol{f}^C - \boldsymbol{f}_2^w\right) \\ \boldsymbol{u}_1^C &= \boldsymbol{u}^C - \left[\boldsymbol{K}_2^C\right]^{-1} \left(\boldsymbol{f}^C - \boldsymbol{f}_2^w\right). \end{aligned} \tag{25}$$

[9] Note that the body to which $\boldsymbol{f}^C, \boldsymbol{f}^w \boldsymbol{u}^C, \boldsymbol{K}^C$ relate to is denoted by a subscript, whereas the body to which individual elements of the vectors $\boldsymbol{f}^C$ and $\boldsymbol{u}^C$ (viz. p_j, q_j, v_j, w_j) relate is denoted by a superscript. This convention has been adopted because $\boldsymbol{f}^C, \boldsymbol{f}^w, \boldsymbol{u}^C, \boldsymbol{K}^C$ already have superscripts, and p_j, q_j, v_j, w_j already denote the node to which they relate with a subscript.

Equation (25) can then be substituted into equation (21a), and simplified as

$$\begin{aligned}
\boldsymbol{f}^C &= \boldsymbol{f}_1^w + \boldsymbol{K}_1^C \boldsymbol{u}_1^C \\
\boldsymbol{f}^C &= \boldsymbol{f}_1^w + \boldsymbol{K}_1^C \left(\boldsymbol{u}^C - \left[\boldsymbol{K}_2^C\right]^{-1} \left(\boldsymbol{f}^C - \boldsymbol{f}_2^w \right) \right) \\
\boldsymbol{f}^C &= \boldsymbol{f}_1^w + \boldsymbol{K}_1^C \boldsymbol{u}^C - \boldsymbol{K}_1^C \left[\boldsymbol{K}_2^C\right]^{-1} \boldsymbol{f}^C + \boldsymbol{K}_1^C \left[\boldsymbol{K}_2^C\right]^{-1} \boldsymbol{f}_2^w \\
\left[\boldsymbol{K}_1^C \left[\boldsymbol{K}_2^C\right]^{-1} + \boldsymbol{I} \right] \boldsymbol{f}^C &= \boldsymbol{f}_1^w + \boldsymbol{K}_1^C \boldsymbol{u}^C + \boldsymbol{K}_1^C \left[\boldsymbol{K}_2^C\right]^{-1} \boldsymbol{f}_2^w \\
\boldsymbol{f}^C &= \left[\boldsymbol{K}_1^C \left[\boldsymbol{K}_2^C\right]^{-1} + \boldsymbol{I} \right]^{-1} \left[\boldsymbol{K}_1^C \boldsymbol{u}^C + \boldsymbol{f}_1^w + \boldsymbol{K}_1^C \left[\boldsymbol{K}_2^C\right]^{-1} \boldsymbol{f}_2^w \right], \quad (26)
\end{aligned}$$

where $\boldsymbol{I}$ is the identity matrix. Equation (26) can be written in the form given in equation (5), which is reproduced here for convenience

$$\boldsymbol{f}^C = \boldsymbol{f}^w + \boldsymbol{K}^C \boldsymbol{u}^C,$$

where

$$\boldsymbol{f}^w = \left[\boldsymbol{K}_1^C \left[\boldsymbol{K}_2^C\right]^{-1} + \boldsymbol{I} \right]^{-1} \left(\boldsymbol{f}_1^w + \boldsymbol{K}_1^C \left[\boldsymbol{K}_2^C\right]^{-1} \boldsymbol{f}_2^w \right) \quad (27a)$$

$$\boldsymbol{K}^C = \left[\boldsymbol{K}_1^C \left[\boldsymbol{K}_2^C\right]^{-1} + \boldsymbol{I} \right]^{-1} \boldsymbol{K}_1^C. \quad (27b)$$

4.5 Repartition the vectors and matrices

Following the static reduction, the A-nodes in the force and displacement column vectors are ordered such that each element alternates between normal and tangential terms, i.e.

$$\boldsymbol{f}^C = \{q_1, p_1, q_2, p_2, q_3, p_3, ..., q_N, p_N\}^T \quad (28a)$$

$$\boldsymbol{u}^C = \{v_1, w_1, v_2, w_2, v_3, w_3, ..., v_N, w_N\}^T, \quad (28b)$$

which also implies that adjacent stiffness values in the reduced contact stiffness matrix $\boldsymbol{K}^C$, alternate between degrees of freedom in the same manner. This can be inconvenient for many applications, and it is usually more straightforward to order the matrices such that all the normal degrees of freedom and all the tangential degrees of freedom are grouped together as

$$\boldsymbol{f}^C = \{q_1, q_2, q_3..., q_N, p_1, p_2, p_3, ..., p_N\}^T = \{\boldsymbol{q}, \boldsymbol{p}\}^T \quad (29a)$$

$$\boldsymbol{u}^C = \{v_1, v_2, v_3, ..., v_N, w_1, w_2, w_3, ..., w_N\}^T = \{\boldsymbol{v}, \boldsymbol{w}\}^T. \quad (29b)$$

Clearly, the terms in the reduced contact stiffness matrix $\boldsymbol{K}^C$, must be moved accordingly. Following this reorganisation, the reduced contact stiffness matrix $\boldsymbol{K}^C$, and can be repartitioned as

$$\boldsymbol{K}^C = \begin{bmatrix} \boldsymbol{A} & \boldsymbol{B}^T \\ \boldsymbol{B} & \boldsymbol{C} \end{bmatrix}, \quad (30)$$

so that the force displacement relation can be written as

$$\begin{Bmatrix} \boldsymbol{q} \\ \boldsymbol{p} \end{Bmatrix} = \begin{bmatrix} \boldsymbol{A} & \boldsymbol{B}^T \\ \boldsymbol{B} & \boldsymbol{C} \end{bmatrix} \begin{Bmatrix} \boldsymbol{v} \\ \boldsymbol{w} \end{Bmatrix} + \begin{Bmatrix} \boldsymbol{q}^w \\ \boldsymbol{p}^w \end{Bmatrix}, \quad (31)$$

where $\boldsymbol{f}^w = \{\boldsymbol{q}^w, \boldsymbol{p}^w\}^T$, and is organised in a similar way to $\boldsymbol{f}^C$.

Note that, if the reader is considering a contact problem between two elastic bodies, the matrices are not to be repartitioned until after having merged the loading vectors and matrices from the two problems, i.e. this repartitioning is not to be done until the final $\boldsymbol{K}^C$ and $\boldsymbol{f}^w$ have been obtained.

5 Static Reduction Procedure: Implementation in MATLAB

In this section, the implementation in MATLAB of the procedure of static reduction outlined in § 4 is described. It is assumed here that the reader has already performed the steps in § 3, and has successfully imported the ABAQUS global stiffness matrix from the `.mtx` file into MATLAB, and has also imported into MATLAB the ABAQUS node sets. Also note that, a MATLAB code that performs the full static reduction procedure on the example model from § 2.1, is provided in § A.9.

5.1 Create reduced stiffness matrix

Once the global stiffness matrix $\boldsymbol{K}$, and the sets of contact, externally loaded, and internal M-nodes have been imported into MATLAB, as described in § 3, the global stiffness matrix can be re-partitioned into several sub-matrices, as in equation (7), which is reproduced here for convenience

$$\begin{Bmatrix} \boldsymbol{f}^I \\ \boldsymbol{f}^E \\ \boldsymbol{f}^C \end{Bmatrix} = \begin{bmatrix} \boldsymbol{K}^{II} & \boldsymbol{K}^{IE} & \boldsymbol{K}^{IC} \\ \boldsymbol{K}^{EI} & \boldsymbol{K}^{EE} & \boldsymbol{K}^{EC} \\ \boldsymbol{K}^{CI} & \boldsymbol{K}^{CE} & \boldsymbol{K}^{CC} \end{bmatrix} \begin{Bmatrix} \boldsymbol{u}^I \\ \boldsymbol{u}^E \\ \boldsymbol{u}^C \end{Bmatrix}.$$

The $\boldsymbol{L}$ matrix may be then calculated using equation (12)

$$\boldsymbol{L} = \left[\begin{bmatrix} \boldsymbol{K}^{EE} & \boldsymbol{K}^{EC} \\ \boldsymbol{K}^{CE} & \boldsymbol{K}^{CC} \end{bmatrix} - \begin{bmatrix} \boldsymbol{K}^{EI} \\ \boldsymbol{K}^{CI} \end{bmatrix} \left[\boldsymbol{K}^{II}\right]^{-1} \begin{bmatrix} \boldsymbol{K}^{IE} & \boldsymbol{K}^{IC} \end{bmatrix} \right],$$

which can subsequently be partitioned, and used to calculate the reduced contact stiffness matrix $\boldsymbol{K}^C$, and also the $\boldsymbol{K}^E$ matrix, which is used to create the load vector $\boldsymbol{f}^w$. For the force controlled case these matrices are calculated as in equation (17)

$$\begin{aligned} \boldsymbol{K}^E &= \boldsymbol{L}^{CE} \left[\boldsymbol{L}^{EE}\right]^{-1} \\ \boldsymbol{K}^C &= \left[\boldsymbol{L}^{CC} - \boldsymbol{L}^{CE} \left[\boldsymbol{L}^{EE}\right]^{-1} \boldsymbol{L}^{EC}\right], \end{aligned}$$

and for the displacement controlled case as in equation (20)

$$\begin{aligned} \boldsymbol{K}^E &= \boldsymbol{L}^{CE} \\ \boldsymbol{K}^C &= \boldsymbol{L}^{CC}. \end{aligned}$$

The code for a MATLAB function that reduces the global stiffness matrix $\boldsymbol{K}$, and outputs the $\boldsymbol{K}^C, \boldsymbol{K}^E, \boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$ matrices is provided in § A.4. This function is named `static_reduction` and has five required inputs: the global stiffness matrix $\boldsymbol{K}$, the sets of internal, externally loaded, and contact M-nodes, and the specification of whether the external load is applied in force (`'force'`) or displacement control (`'displ'`). This function then outputs the $\boldsymbol{K}^C$,

$\boldsymbol{K}^E$, $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$ matrices, in that order, i.e.

```
[KC, KE, A, B, C] = static_reduction(K, ...
    internal_nodes, ext_loaded_nodes, contact_nodes, 'force')
```

for the force control case, and

```
[KC, KE, A, B, C] = static_reduction(K, ...
    internal_nodes, ext_loaded_nodes, contact_nodes, 'displ')
```

for the displacement controlled case, where the $\boldsymbol{K}^C$ and $\boldsymbol{K}^E$ have not been repartitioned and are organised to be compatible with $\boldsymbol{f}^C$ and $\boldsymbol{u}^E$ as in equation (28)

$$\begin{aligned}\boldsymbol{f}^C &= \{q_1, p_1, q_2, p_2, q_3, p_3, ..., q_N, p_N\}^T \\ \boldsymbol{u}^C &= \{v_1, w_1, v_2, w_2, v_3, w_3, ..., v_N, w_N\}^T,\end{aligned}$$

and the $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$ matrices are the repartitioned sub-matrices organised to be compatible with $\boldsymbol{f}^C$ and $\boldsymbol{u}^E$ as in equation (29).

$$\begin{aligned}\boldsymbol{f}^C &= \{q_1, q_2, q_3..., q_N, p_1, p_2, p_3, ..., p_N\}^T = \{\boldsymbol{q}, \boldsymbol{p}\}^T \\ \boldsymbol{u}^C &= \{v_1, v_2, v_3, ..., v_N, w_1, w_2, w_3, ..., w_N\}^T = \{\boldsymbol{v}, \boldsymbol{w}\}^T.\end{aligned}$$

The $\boldsymbol{K}^E$ matrix is to be used to create the load vector $\boldsymbol{f}^w$. The $\boldsymbol{K}^C$ matrix is provided for the case when the contact between two elastic bodies is to be considered and the two $\boldsymbol{K}^C$ matrices must be merged with another matrix prior to repartitioning. If the problem to be considered is between a single elastic body in contact with a rigid surface, then the $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$ matrices output by this function can be used and the $\boldsymbol{K}^C$ matrix may not be required.

5.2 Create load vectors

The force displacement relation for the contact M-nodes can be written as in equation (5), which is reproduced here for convenience

$$\boldsymbol{f}^C = \boldsymbol{f}^w + \boldsymbol{K}^C \boldsymbol{u}^C,$$

where the loading vector $\boldsymbol{f}^w$ is computed as in equation (16)

$$\boldsymbol{f}^w = \boldsymbol{K}^E \boldsymbol{f}^E,$$

for the force controlled case, and as in equation (19)

$$\boldsymbol{f}^w = \boldsymbol{K}^E \boldsymbol{u}^E,$$

for the displacement controlled case.

The code for a MATLAB function that creates the load vector $\boldsymbol{f}^w$, is provided in § A.5. This function is named `create_load_vector` and has three input parameters: the $\boldsymbol{K}^E$ matrix, specification of the distribution of load that is to be applied to the M-nodes of degree of freedom 1, and specification of the distribution of load that is to be applied to the M-nodes of degree of freedom 2, as shown below.

```
fw_unsorted = create_load_vector(KE, load_distribution_DOF_one, ...
    load_distribution_DOF_two)
```

Note that, at the current level of sophistication, the function `create_load_vector` can **not** create a load vector $\boldsymbol{f}^w$ from a *Node set* that only includes M-nodes of one degree of freedom. Therefore, the first input parameter of the function, the $\boldsymbol{K}^E$ matrix, **must** be created from a set of externally loaded nodes that is imported using the option `'both'` as the fourth input parameter in the function `import_node_set`. This function can be improved to remove this limitation in subsequent revisions of the code.

There are several optional load distributions, which are

1. `'zero'`: all values equal to zero.
2. `'constant'`: all values equal to +1.
3. `'-constant'`: all values equal to -1.
4. `'linear'`: linear variation from -1 at the first element to +1 at the last element.
5. `'-linear'`: linear variation from +1 at the first element to -1 at the last element.
6. `'pmstep'`: first half of the elements equal to +1 and the second half of the elements equal to -1 (will not work with an odd number of externally loaded nodes).
7. `'mpstep'`: first half of the elements equal to -1 and the second half of the elements equal to +1 (will not work with an odd number of externally loaded nodes).

One of these options must be input for degree of freedom 1 and another for degree of freedom 2, although in many cases one of the distributions will be `'zero'`. Whether the problem is force controlled or displacement controlled does not enter into this function, and is determined by the $\boldsymbol{K}^E$ matrix, which results from the static reduction in § 5.1. If the problem is force controlled, the the load distribution represents $\boldsymbol{f}^E$, and if the problem is displacement controlled then the load distribution represents $\boldsymbol{u}^E$.

This function then outputs the $\boldsymbol{f}^w$ load vector corresponding to the external load that was specified, which is partitioned to be compatible with the original the force and displacement vectors organised as in equation (28)

$$\begin{aligned}\boldsymbol{f}^C &= \{q_1, p_1, q_2, p_2, q_3, p_3, ..., q_N, p_N\}^T \\ \boldsymbol{u}^C &= \{v_1, w_1, v_2, w_2, v_3, w_3, ..., v_N, w_N\}^T .\end{aligned}$$

This is how the load vector $\boldsymbol{f}^w$ must be organised if it is to be further manipulated to be used in a contact problem between two elastic bodies, as is described in § 5.5. If, however, contact between a single elastic body and a rigid surface is to be considered, then this vector must be reorganised so as to be compatible with the $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$ matrices using the MATLAB function that is described in § 5.3.

For most problems, there will be more than one applied load (i.e. a normal load, a shear load, and a bulk load). In these cases, this function can be called several times to create a load vector $\boldsymbol{f}^w$ corresponding to each applied load. These loads can then be scaled to reflect the loading history of the problem that is being considered.

5.3 Reorganise load vector

The load vector $\boldsymbol{f}^w$ created in § 5.2 is organised to be compatible with the $\boldsymbol{K}^C$ matrix and has its elements as in equation (28). In order to have the load vector $\boldsymbol{f}^w$ be compatible with the $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$ matrices it must be organised as in equation (29), which is reproduced here for convenience.

$$\begin{aligned}\boldsymbol{f}^C &= \{q_1, q_2, q_3..., q_N, p_1, p_2, p_3, ..., p_N\}^T = \{\boldsymbol{q}, \boldsymbol{p}\}^T \\ \boldsymbol{u}^C &= \{v_1, v_2, v_3, ..., v_N, w_1, w_2, w_3, ..., w_N\}^T = \{\boldsymbol{v}, \boldsymbol{w}\}^T .\end{aligned}$$

The code for a MATLAB function that reorganises the load vector $\boldsymbol{f}^w$, to be compatible with the $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$ matrices, is provided in § A.6. This function is named `resort_load_vector` and has only one input: a load vector $\boldsymbol{f}^w$ that has been created using the `create_load_vector` function described in § 5.2. The `resort_load_vector` function then outputs the reorganised load vector, as shown below.

```
fw_organised = resort_load_vector(fw_unsorted)
```

5.4 Create $\boldsymbol{K}^C$ for contact between two elastic bodies

To create the reduced contact stiffness matrix $\boldsymbol{K}^C$ for contact between two elastic bodies, first the reduced matrices must be computed separately for each body in contact with a rigid plane obstacle. These matrices must then be merged as in equation (27b), which is reproduced here for convenience.

$$\boldsymbol{K}^C = \left[\boldsymbol{K}_1^C \left[\boldsymbol{K}_2^C\right]^{-1} + \boldsymbol{I}\right]^{-1} \boldsymbol{K}_1^C.$$

The code for a MATLAB function that merges two $\boldsymbol{K}^C$ matrices and creates a new $\boldsymbol{K}^C$ matrix for the case of these two elastic bodies in contact, is provided in § A.7. This function is named `merge_matrices` and has two input parameters: the two $\boldsymbol{K}^C$ matrices that are to be merged. The $\boldsymbol{K}^C$ matrix designated as `body_two` must **not** have rigid body degrees of freedom. Also note that, the two $\boldsymbol{K}^C$ matrices to be merged must have the same number of A-nodes, distributed in the same way, and in the same sequence.

The `merge_matrices` function then outputs the reduced contact stiffness matrix $\boldsymbol{K}^C$, for the two bodies in contact, as well as the associated $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$ matrices, as shown below.

```
[KC, A, B, C] = merge_matrices(body_one_KC, body_two_KC)
```

Note that the $\boldsymbol{K}^C$ matrix (`KC`) is organised to be compatible with the force and displacement vectors as in equation (28), and the $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$ matrices (`A, B, C`) are the repartitioned submatrices of this new $\boldsymbol{K}^C$ matrix.

5.5 Create $\boldsymbol{f}^w$ for contact between two elastic bodies

To create the load vector $\boldsymbol{f}^w$ for contact between two elastic bodies, first the load vectors must be computed separately for each body in contact with a rigid plane obstacle. These load vectors can then be manipulated as in equation (27b)

$$\boldsymbol{f}^w = \left[\boldsymbol{K}_1^C \left[\boldsymbol{K}_2^C\right]^{-1} + \boldsymbol{I}\right]^{-1} \left(\boldsymbol{f}_1^w + \boldsymbol{K}_1^C \left[\boldsymbol{K}_2^C\right]^{-1} \boldsymbol{f}_2^w\right)$$

for the case when contact between these two elastic bodies is considered.

The code for a MATLAB function that creates the load vector $\boldsymbol{f}^w$ for the case when two elastic bodies are in contact is provided in § A.8. This function is named `create_two_body_load_vector` and the required inputs are the two $\boldsymbol{K}^C$ matrices that were merged to create the new reduced stiffness matrix for the two body problem (using the MATLAB function described in § 5.4), and the load vector $\boldsymbol{f}^w$ that is to be manipulated for use in the two body problem. The load vector $\boldsymbol{f}^w$ that is input to this function, must **not be reorganised** to be compatible with the $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$ matrices; it must be compatible with the $\boldsymbol{K}^C$ matrix format. In almost all cases, the load vector $\boldsymbol{f}^w$, from only one body should be input into the function, and where the load vector for the other body would be input, the string `'none'` should be specified instead, as shown below.

```
fw_unsorted_1 = create_two_body_load_vector(body_one_KC, ...
    body_two_KC, body_one_fw_unsorted, 'none')
fw_unsorted_2 = create_two_body_load_vector(body_one_KC, ...
    body_two_KC, 'none', body_two_fw_unsorted)
```

The output of this function is a new load vector $\boldsymbol{f}^w$, that corresponds to the case of the contact between the two elastic bodies whose reduced contact stiffness matrices $\boldsymbol{K}^C$ are input. Note that, this new load vector must be reorganised as described in § 5.3 to be compatible with the $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$ matrices. As in § 5.2, this function can be called several times to create one load vector $\boldsymbol{f}^w$, corresponding to each applied load for the case of contact between two elastic bodies. These load vectors $\boldsymbol{f}^w$ can then be scaled to reflect the loading history of the problem that is being considered.

6 Conclusions

The theory and implementation of a procedure for the static reduction of two dimensional, linear elastic frictional contact problems has been described. The result of this procedure, the reduced contact stiffness matrix, can be used for several analyses; this includes solving the transient 'marching in time' problem, given a particular loading history. Preliminary results indicate that this method of solution runs approximately an order of magnitude more quickly than does the equivalent problem in ABAQUS. However, the relative efficiency of solving the transient problem using static reduction vs. ABAQUS is problem dependent. It is reasonable to expect the improvement in efficiency achieved by the static reduction technique to increase with the ratio of internal nodes to contact nodes, but this has not yet been verified.

The reduced contact stiffness matrix can be used for other analyses, for example to determine the frictional shakedown limit, i.e. the magnitude of loading above which the steady state will be dissipative, irrespective of the initial conditions. The shakedown limit can be determined exactly using a numerical optimisation technique with a relatively small amount of computational effort. The reduced contact stiffness matrix may also allow for the the degree to which the direct and shearing tractions are coupled to be quantified, which would enable an estimate to be made of the range of load over which the steady state is dependant on initial conditions.

A Additional Material

In this section, supplemental material that is mentioned in the previous sections is provided. Note that, when copying and pasting code into a MATLAB `.m` file, some characters may not copy correctly. A character for which this commonly occurs is: `'`. Generally the error in transcription is consistent, and so the *Find & Replace* function in MATLAB can be used to remedy the error. Errors also occasionally occur when copying the character: _.

A.1 The *Input file* for the ABAQUS model from § 2.1

```
*Heading
** Job name: example Model name: Model-1
** Generated by: ABAQUS Student Edition 6.11-2
*Preprint, echo=NO, model=NO, history=NO, contact=NO
**
** PARTS
**
*Part, name=RR_example
*Node
      1,           1.,           1.
      2,           0.,           1.
      3,          -1.,           1.
      4,           1.,           0.
      5,           0.,           0.
      6,          -1.,           0.
      7,           1.,          -1.
      8,           0.,          -1.
      9,          -1.,          -1.
*Element, type=CPS4R
1, 1, 2, 5, 4
2, 2, 3, 6, 5
3, 4, 5, 8, 7
4, 5, 6, 9, 8
*Nset, nset=_PickedSet2, internal, generate
 1,  9,  1
*Elset, elset=_PickedSet2, internal, generate
 1,  4,  1
*Nset, nset=aa-contact, generate
 7,  9,  1
*Nset, nset=b-ext_loaded, generate
 1,  3,  1
*Nset, nset=c-internal, generate
 4,  6,  1
** Section: Section-1
*Solid Section, elset=_PickedSet2, material=Material-1
,
*End Part
**
**
** ASSEMBLY
**
*Assembly, name=Assembly
**
```

```
*Instance, name=RR_example-1, part=RR_example
*End Instance
**
*End Assembly
**
** MATERIALS
**
*Material, name=Material-1
*Elastic
 2e+11, 0.3
** ----------------------------------------------------------------
**
** STEP: Step-1
**
*Step, name=Step-1
*Static
1., 1., 1e-05, 1.
**
** OUTPUT REQUESTS
**
*Restart, write, frequency=0
**
** FIELD OUTPUT: F-Output-1
**
*Output, field, variable=PRESELECT
**
** HISTORY OUTPUT: H-Output-1
**
*Output, history, variable=PRESELECT
*End Step
```

A.2 MATLAB function to import a `.mtx` file from § 3.3

To create a MATLAB function that imports a `.mtx` file into a MATLAB sparse matrix, copy the following code into a new MATLAB editor `.m` file, and save it as `import_stiffness_matrix.m`.

```
function [matlab_stiffness_matrix] = import_stiffness_matrix(mtx_file)
%============== Import Stiffness Matrix ==============%
abaqus_stiffness_matrix = dlmread(mtx_file);
% merge node number info from column 1 and DOF info from column 2 and
% store in the 1st column of a new matrix
matlab_nodes(:,1) = 2*(abaqus_stiffness_matrix(:,1)-1)+ ...
    abaqus_stiffness_matrix(:,2);
% merge node number info from column 3 and DOF info from column 4 and
% store in the 2nd column of a new matrix
matlab_nodes(:,2) = 2*(abaqus_stiffness_matrix(:,3)-1)+ ...
    abaqus_stiffness_matrix(:,4);
% extract the stiffness values from the .mtx file, and store in a double
% length vector
stiffness_values = [abaqus_stiffness_matrix(:,5); ...
    abaqus_stiffness_matrix(:,5)];
% create a matrix of the new matlab node numbers, and a vector of indices
% of their position in the abaqus stiffness matrix
[matlab_matrix_indices, abaqus_stiffness_value_index] = unique( ...
    [matlab_nodes; matlab_nodes(:,2) matlab_nodes(:,1)], 'rows');
% compile the stiffness matrix using the new node numbering convention
matlab_stiffness_matrix = accumarray( matlab_matrix_indices, ...
    stiffness_values(abaqus_stiffness_value_index), [], @max, [], true);
```

A.3 MATLAB function to import the ABAQUS node sets from § 3.5

To create a MATLAB function that imports *Node sets* into MATLAB from an ABAQUS *Input file* file, copy the following code into a new MATLAB editor `.m` file, and save it as `import_node_set.m`.

```
function [matlab_node_set, abaqus_node_set] = ...
    import_node_set(input_file, nodeset_name, ...
    input_file_length, DOF_to_import, sorting_DOF, sorting_order)
%================== Import Node Set ==================%
% find line that contains the name of the node set to be imported
find_line = ['*Nset, nset=',nodeset_name];
i = 0; count = 0;
read_input_file = fopen(input_file,'r');
while i == 0
    count = count +1;
    current_line = fgetl(read_input_file);
    i = strcmp(current_line,find_line);
    if count > input_file_length, break, end
end
% if found, import the node set
if count < input_file_length
    tempvar = textscan(read_input_file,'%n','delimiter',',');
    node_set = tempvar{1}';
    fclose(read_input_file);
% if not found see if the command ', generate' is after the node set
elseif count > input_file_length
    i = 0; count =0; fclose(read_input_file);
    read_input_file = fopen(input_file,'r');
    find_line = ['*Nset, nset=',nodeset_name,', generate'];
    while i == 0
        count = count +1;
        current_line = fgetl(read_input_file);
        i = strcmp(current_line,find_line);
        if count > input_file_length, break, end
    end
%     if still not found generate an error message
    if count > input_file_length
        error('Error: cannot find nodeset in input file.')
    end
%     if found once ', generate' is added, import the node set
    temp_textscan_output = textscan(read_input_file,'%d',...
        'delimiter',',');
    node_info = temp_textscan_output{1}';
    node_set = zeros(1,1+(node_info(1,2)-...
        node_info(1,1))/node_info(1,3));
    for i = 1:length(node_set)
        node_set(1,i) = node_info(1,1) + ...
            (i-1)*(node_info(1,3));
    end
    fclose(read_input_file);
end
```

```
%================== Sort Node Set ==================%
% check if sorting inputs exist, and sort the node set if they do
if exist('sorting_DOF','var') == 1
%     find line where node number and position coordinates are listed
    find_line = '*Node';
    i = 0; count = 0;
    read_input_file = fopen(input_file,'r');
    while i == 0
        count = count +1;
        current_line = fgetl(read_input_file);
        i = strcmp(current_line,find_line);
        if count > input_file_length, break, end
    end
    if count > input_file_length
        error('Error: cannot find line ''*Node'' in input file.')
    end
%     extract numbers and posistions of all nodes in the input file
    tempvar = textscan(read_input_file,'%n, %n, %n');
    all_nodes = tempvar{1};
    node_positions = [tempvar{2},tempvar{3}];
%     extract position coordinates of only the nodes in the node set
    info_list = zeros(length(node_set),3);
    j = 1;
    for i = 1:node_set(length(node_set))
        if node_set(j,1) == all_nodes(i);
            info_list(j,:) = [i,node_positions(i,:)];
            j=j+1;
        end
    end
%     define which coordinate direction to sort with respect to
    if sorting_DOF == 1
        sort_column = 2;
    elseif sorting_DOF == 2
        sort_column = 3;
    else
        error('Error: incorrect input option selected for ''sorting_DOF''')
    end
%     sort the node set in ascending or descending order
    if strcmp(sorting_order,'ascending')
        ordered_info_list = sortrows(info_list,sort_column);
        abaqus_node_set = ordered_info_list(:,1)';
    elseif strcmp(sorting_order,'descending')
        ordered_info_list = sortrows(info_list,-sort_column);
        abaqus_node_set = ordered_info_list(:,1)';
    else
        error('Error: incorrect input option selected for ''sorting_order''')
    end
% check if sorting inputs exist, and don't sort the node set if they don't
elseif exist('sorting_DOF','var') == 0
    abaqus_node_set = node_set;
else
    error('Error: error trying to sort the node set')
end
```

```
%============ Convert A-Nodes to M-nodes =============%
if exist('DOF_to_import','var')==0
    DOF_to_import = 'both';
end
if DOF_to_import == 1
    matlab_node_set = zeros(1,length(abaqus_node_set));
    for j = 1:length(abaqus_node_set)
        matlab_node_set(1,j) = 2.*(abaqus_node_set(1,j)-1)+1;
    end
elseif DOF_to_import == 2
    matlab_node_set = zeros(1,length(abaqus_node_set));
    for j = 1:length(abaqus_node_set)
        matlab_node_set(1,j) = 2.*(abaqus_node_set(1,j)-1)+2;
    end
elseif strcmp(DOF_to_import,'both') == 1
    matlab_node_set = zeros(1,2*length(abaqus_node_set));
    for j = 1:length(abaqus_node_set)
        matlab_node_set(1, 2*(j-1)+1) = 2.*(abaqus_node_set(1,j)-1)+1;
        matlab_node_set(1, 2*(j-1)+2) = 2.*(abaqus_node_set(1,j)-1)+2;
    end
else
    error('Error: incorrect input option selected ''DOF_to_import''')
end
```

A.4 MATLAB function to create K^C, K^E, A, B, C from § 5.1

To create a MATLAB function that performs the static reduction of the global stiffness matrix, copy the following code into a `.m` file, and save the script as `static_reduction.m`.

```
function [KC, KE, A, B, C] = static_reduction(K, ...
    internal_nodes, ext_loaded_nodes, contact_nodes, load_type)
%================= Static Reduction =================%
% create the submatrices of the global stiffness matrix (K)
KII = K(internal_nodes, internal_nodes);
KIE = K(internal_nodes, ext_loaded_nodes);
KEI = KIE';
KIC = K(internal_nodes, contact_nodes);
KCI = KIC';
KEE = K(ext_loaded_nodes, ext_loaded_nodes);
KEC = K(ext_loaded_nodes, contact_nodes);
KCE = KEC';
KCC = K(contact_nodes, contact_nodes);
% create the L matrix and submatrices using the submatrices of K
L = [KEE,KEC;KCE,KCC]-[KEI;KCI]*(KII\[KIE,KIC]);
LEE = L(1:length(KEE(:,1)), 1:length(KEE(:,1)));
LEC = L(1:length(KEE(:,1)), length(KEE(:,1))+1:length(L(:,1)));
LCE = LEC';
LCC = L(length(KEE(:,1))+1:length(L(:,1)), ...
    length(KEE(:,1))+1:length(L(:,1)));
%============== Make KC and KE Matrices ==============%
if strcmp(load_type,'force') == 1
    KC = full(LCC-LCE*(LEE\LEC));
    KE = full(LCE/LEE);
elseif strcmp(load_type,'displ') == 1
    KC = full(LCC);
    KE = full(LCE);
else
    error('Error: incorrect load_type selected. Must be ''force'' or ''displ''.')
end
%=============== Make A, B, C Matrices ===============%
degree_of_freedom_one = zeros(1,length(KC)/2);
degree_of_freedom_two = zeros(1,length(KC)/2);
for j = 1:length(KC)/2
    degree_of_freedom_one(1,j) = 2*(j-1)+1;
    degree_of_freedom_two(1,j) = 2*(j-1)+2;
end
A = KC(degree_of_freedom_one, degree_of_freedom_one);
B = KC(degree_of_freedom_two, degree_of_freedom_one);
C = KC(degree_of_freedom_two, degree_of_freedom_two);
```

A.5 MATLAB function to create $\boldsymbol{f}^w$ from § 5.2

To create a MATLAB function that creates a load vector $\boldsymbol{f}^w$, copy the following code into a `.m` file, and save the script as `create_load_vector.m`.

```
function [fw_unsorted] = create_load_vector( KE, ...
    load_distribution_DOF_one, load_distribution_DOF_two)
%============= Create Load Distributions =============%
load_zero = zeros(length(KE(1,:))/2,1);
load_constant = ones(length(KE(1,:))/2,1);
load_linear = zeros(length(KE(1,:))/2,1); %preallocate
for i = 0:length(load_linear)-1
   load_linear(i+1,1) = -1 + 2*(i/(length(load_linear)-1));
end
%========= Select Load Distribution for DOF 1 ========%
if strcmp(load_distribution_DOF_one,'zero') == 1;
    load_distribution_DOF_one = load_zero;
elseif strcmp(load_distribution_DOF_one,'constant') == 1;
    load_distribution_DOF_one = load_constant;
elseif strcmp(load_distribution_DOF_one,'-constant') == 1;
    load_distribution_DOF_one = -load_constant;
elseif strcmp(load_distribution_DOF_one,'linear') == 1;
    load_distribution_DOF_one = load_linear;
elseif strcmp(load_distribution_DOF_one,'-linear') == 1;
    load_distribution_DOF_one = -load_linear;
elseif strcmp(load_distribution_DOF_one,'pmstep') == 1;
    load_pmstep = [ones(length(KE(1,:))/4,1); ...
        -ones(length(KE(1,:))/4,1)];
    load_distribution_DOF_one = load_pmstep;
elseif strcmp(load_distribution_DOF_one,'mpstep') == 1;
    load_mpstep = [-ones(length(KE(1,:))/4,1); ...
        ones(length(KE(1,:))/4,1)];
    load_distribution_DOF_one = load_mpstep;
else
    error('Error: incorrect load distribution selected for DOF1')
end
%========= Select Load Distribution for DOF 2 ========%
if strcmp(load_distribution_DOF_two,'zero') == 1;
    load_distribution_DOF_two = load_zero;
elseif strcmp(load_distribution_DOF_two,'constant') == 1;
    load_distribution_DOF_two = load_constant;
elseif strcmp(load_distribution_DOF_two,'-constant') == 1;
    load_distribution_DOF_two = -load_constant;
elseif strcmp(load_distribution_DOF_two,'linear') == 1;
    load_distribution_DOF_two = load_linear;
elseif strcmp(load_distribution_DOF_two,'-linear') == 1;
    load_distribution_DOF_two = -load_linear;
elseif strcmp(load_distribution_DOF_two,'pmstep') == 1;
    load_pmstep = [ones(length(KE(1,:))/4,1); ...
        -ones(length(KE(1,:))/4,1)];
    load_distribution_DOF_two = load_pmstep;
elseif strcmp(load_distribution_DOF_two,'mpstep') == 1;
    load_mpstep = [-ones(length(KE(1,:))/4,1); ...
        ones(length(KE(1,:))/4,1)];
    load_distribution_DOF_two = load_mpstep;
```

```
else
    error('Error: incorrect load distribution selected for DOF2')
end
%==================== Create fw ====================%
load_distribution = zeros(2*length(load_distribution_DOF_one),1);
for j = 1:(length(KE(1,:))/2)
    load_distribution(2*(j-1)+1, 1) = load_distribution_DOF_one(j,1);
    load_distribution(2*(j-1)+2, 1) = load_distribution_DOF_two(j,1);
end
fw_unsorted = KE*load_distribution;
```

A.6 MATLAB function to reorganise $\boldsymbol{f}^w$ from §5.3

To create a MATLAB function that reorganises the load vector $\boldsymbol{f}^w$ so that it is compatible with the $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$ matrices, copy the following code into a `.m` file, and save the script as `resort_load_vector.m`.

```
function [load_vector] = resort_load_vector(fw_load_vector)
%================= Re-sort Node Set ==================%
load_vector = zeros(length(fw_load_vector), 1);
% create a vector of even rows and of odd rows
for j = 1:length(load_vector)/2
    load_vector(j, 1) = fw_load_vector(2*(j-1)+1, 1);
    load_vector(length(load_vector)/2+j, 1) = fw_load_vector(2*(j-1)+2, 1);
end
```

A.7 MATLAB function to combine $\boldsymbol{K}^C$ matrices from § 5.4

To create a MATLAB function that combines two $\boldsymbol{K}^C$ matrices to produce the $\boldsymbol{K}^C$ matrix for the contact of those two elastic bodies, copy the following code into a `.m` file, and save the script as `merge_matrices.m`.

```
function [KC, A, B, C] = merge_matrices(body_one_KC, body_two_KC)
%================ Merge KC Matrices ================%
KC = (body_one_KC/body_two_KC + eye(size(body_two_KC)))\body_one_KC;
%============== Create A, B, C Matrices ==============%
degree_of_freedom_one = zeros(1,length(KC)/2);
degree_of_freedom_two = zeros(1,length(KC)/2);
for j = 1:length(KC)/2
    degree_of_freedom_one(1,j) = 2*(j-1)+1;
    degree_of_freedom_two(1,j) = 2*(j-1)+2;
end
A = KC(degree_of_freedom_one, degree_of_freedom_one);
B = KC(degree_of_freedom_two, degree_of_freedom_one);
C = KC(degree_of_freedom_two, degree_of_freedom_two);
```

A.8 MATLAB function to create $\boldsymbol{f}^w$ for two body problems from § 5.5

To create a MATLAB function that manipulate the load vector $\boldsymbol{f}^w$, for the problem of contact between two elastic bodies, copy the following code into a `.m` file, and save the script as `create_two_body_load_vector.m`.

```
function [load_vector] = create_two_body_load_vector( ...
    body_one_KC, body_two_KC, body_one_fw, body_two_fw)
%========== Create fw For Two Elastic Bodies =========%
if strcmp(body_one_fw,'none') == 1;
    body_one_fw = zeros(length(body_one_KC(:,1)),1);
elseif strcmp(body_two_fw,'none') == 1;
    body_two_fw = zeros(length(body_two_KC(:,1)),1);
else
end
load_vector = (body_one_KC/body_two_KC + eye(size(body_two_KC)))\ ...
     (body_one_fw + body_one_KC*(body_two_KC\body_two_fw));
```

A.9 Static reduction of the example model from § 2.1

A MATLAB code is provided below that performs the static reduction and creates the $\boldsymbol{K}^C, \boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$ matrices and the $\boldsymbol{f}^w$ loading vector for a pressure applied to the top of the example model created in § 2.1. First the case of a downward pressure applied to the top of an elastic square in contact with a rigid plane obstacle is considered. Then the elastic-elastic contact between the faces of two elastically similar blocks is considered. The edge of body 1 is loaded in force control creating a compressive contact load, while the far edge of the second body is constrained to have zero displacement in both direction 1 and direction 2.

```
% define file paths
input_file = 'example.inp';
mtx_file = 'example-1_STIF1.mtx';
% import node sets
[Mnodes_internal, Anodes_internal] = ...
    import_node_set(input_file, 'c-internal', 72);
[Mnodes_ext_loaded, Anodes_ext_loaded] = ...
    import_node_set(input_file, 'b-ext_loaded', 72);
[Mnodes_contact_body1, Anodes_contact_body1] = ...
    import_node_set(input_file, 'aa-contact', 72, 'both', ...
    1, 'ascending');
[Mnodes_contact_body2, Anodes_contact_body2] = ...
    import_node_set(input_file, 'aa-contact', 72, 'both', ...
    1, 'descending');
% combine node sets
contact_Mnodes = Mnodes_contact_body1;
ext_loaded_Mnodes = Mnodes_ext_loaded;
internal_Mnodes = Mnodes_internal;
% import stiffness matrix
K = import_stiffness_matrix(mtx_file);

%====== Static Reduction: Elastic Block on Rigid =====%
[KC, KE, A, B, C] = static_reduction(K, internal_Mnodes, ...
    ext_loaded_Mnodes, contact_Mnodes, 'force');
% create fw load vector, and resort load vector to be compatible with
% the A, B, C matrices
fw_pressure_unsorted = create_load_vector(KE, 'zero', '-constant');
fw_pressure = resort_load_vector(fw_pressure_unsorted);

%======== Static Reduction: Elastic - Elastic =======%
[KC_force, KE_force] = static_reduction(K, internal_Mnodes, ...
    ext_loaded_Mnodes, Mnodes_contact_body1, 'force');
[KC_displ, KE_displ] = static_reduction(K, internal_Mnodes, ...
    ext_loaded_Mnodes, Mnodes_contact_body2, 'displ');
% create load vector
fw_pressure_unsorted = create_load_vector(KE, 'zero', '-constant');
fw_pressure = resort_load_vector(fw_pressure_unsorted);
% merge matrices
body_one_KC = KC_force;
body_two_KC = KC_displ;
[twobody_KC, twobody_A, twobody_B, twobody_C] = ...
    merge_matrices(body_one_KC, body_two_KC);
twobody_fw_pressure = create_two_body_load_vector( ...
    body_one_KC, body_two_KC, fw_pressure_unsorted, 'none');
```